Credit Card Fraud Detection using Python

- Language: R or Python
- Data set: Data on the transaction of credit cards is used here as a data set.

Credit card fraud is a pervasive issue, and its incidence has been escalating in recent years. With projections indicating that we may surpass a billion credit card users by the end of 2022, the need for robust fraud detection mechanisms is more critical than ever. Fortunately, advancements in technologies such as artificial intelligence, machine learning, and data science have empowered credit card companies to detect and prevent fraudulent activities with impressive accuracy.

The core idea behind these advanced detection systems is to analyze the usual spending behaviors of customers, including the geographical locations of their transactions, to distinguish between legitimate and fraudulent activities. By leveraging sophisticated algorithms and models, these systems can learn to identify anomalies that signify potential fraud.

For this project, you can choose either R or Python to work with a dataset containing customers' transaction histories. The goal is to build models using decision trees, artificial neural networks (ANNs), and logistic regression to classify transactions as fraudulent or non-fraudulent. By continuously feeding new data into these models, you can enhance their accuracy and reliability over time.

Steps to Approach the Project:

1. **Data Collection and Preprocessing:**
    - Gather a comprehensive dataset of credit card transactions, which includes features such as transaction amount, transaction location, time of transaction, and customer details.
    - Clean the dataset to handle missing values, outliers, and any inconsistencies.
    - Feature engineering: Create new features that could help improve model performance, such as the frequency of transactions,

average transaction amount, and deviation from the customer's typical transaction locations.

2. **Exploratory Data Analysis (EDA):**
    - Visualize the data to understand the distribution of features and identify any patterns or anomalies.
    - Use statistical methods to explore correlations between different features and the target variable (fraudulent vs. non-fraudulent).
3. **Model Building:**
    - **Decision Trees:** Build a decision tree model to classify transactions. Fine-tune the hyperparameters to improve accuracy.
    - **Artificial Neural Networks:** Develop an ANN model for classification. Experiment with different architectures, activation functions, and optimization algorithms.
    - **Logistic Regression:** Implement a logistic regression model. Regularize the model to prevent overfitting and improve generalization.
4. **Model Training and Evaluation:**
    - Split the data into training and testing sets to evaluate model performance.
    - Use metrics such as accuracy, precision, recall, F1-score, and ROC-AUC to assess the models.
    - Perform cross-validation to ensure the models' robustness.
5. **Model Tuning and Optimization:**
    - Experiment with different techniques to enhance model performance, such as grid search for hyperparameter tuning, feature scaling, and ensemble methods.
    - Continuously update the models with new transaction data to keep them current and improve their accuracy.
6. **Implementation and Monitoring:**
    - Deploy the models into a real-time system to monitor and flag suspicious transactions.
    - Set up an alert system to notify relevant authorities or customers in case of detected fraud.
    - Regularly monitor the models' performance and retrain them as necessary to maintain their effectiveness.

- Here is the overall code snippet and outcomes:

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
sns.set()
%matplotlib inline

df = pd.read_csv('creditcard.csv')
print(df.shape)
df.head()
```

```
(284807, 31)
```

Out[1]:

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V21	V22	V23	V24	V25
0	0.0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698	0.363787	...	-0.018307	0.277838	-0.110474	0.066928	0.12853
1	0.0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102	-0.255425	...	-0.225775	-0.638672	0.101288	-0.339846	0.16717
2	1.0	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676	-1.514654	...	0.247998	0.771679	0.909412	-0.689281	-0.32764
3	1.0	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436	-1.387024	...	-0.108300	0.005274	-0.190321	-1.175575	0.64737
4	2.0	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533	0.817739	...	-0.009431	0.798278	-0.137458	0.141267	-0.20601

5 rows × 31 columns

In [2]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 284807 entries, 0 to 284806
Data columns (total 31 columns):
 #   Column  Non-Null Count   Dtype  
---  ------  --------------   -----  
 0   Time    284807 non-null  float64
 1   V1      284807 non-null  float64
 2   V2      284807 non-null  float64
 3   V3      284807 non-null  float64
 4   V4      284807 non-null  float64
 5   V5      284807 non-null  float64
 6   V6      284807 non-null  float64
 7   V7      284807 non-null  float64
 8   V8      284807 non-null  float64
 9   V9      284807 non-null  float64
 10  V10     284807 non-null  float64
 11  V11     284807 non-null  float64
 12  V12     284807 non-null  float64
 13  V13     284807 non-null  float64
 14  V14     284807 non-null  float64
 15  V15     284807 non-null  float64
 16  V16     284807 non-null  float64
 17  V17     284807 non-null  float64
 18  V18     284807 non-null  float64
 19  V19     284807 non-null  float64
 20  V20     284807 non-null  float64
```

```
 25  V25     284807 non-null  float64
 26  V26     284807 non-null  float64
 27  V27     284807 non-null  float64
 28  V28     284807 non-null  float64
 29  Amount  284807 non-null  float64
 30  Class   284807 non-null  int64
dtypes: float64(30), int64(1)
memory usage: 67.4 MB
```

In [3]:
```
df.describe()
```

Out[3]:

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	..
count	284807.000000	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	..
mean	94813.859575	3.918649e-15	5.682686e-16	-8.761736e-15	2.811118e-15	-1.552103e-15	2.040130e-15	-1.698953e-15	-1.893285e-16	-3.147640e-15	..
std	47488.145955	1.958696e+00	1.651309e+00	1.516255e+00	1.415869e+00	1.380247e+00	1.332271e+00	1.237094e+00	1.194353e+00	1.098632e+00	..
min	0.000000	-5.640751e+01	-7.271573e+01	-4.832559e+01	-5.683171e+00	-1.137433e+02	-2.616051e+01	-4.355724e+01	-7.321672e+01	-1.343407e+01	..
25%	54201.500000	-9.203734e-01	-5.985499e-01	-8.903648e-01	-8.486401e-01	-6.915971e-01	-7.682956e-01	-5.540759e-01	-2.086297e-01	-6.430976e-01	..
50%	84692.000000	1.810880e-02	6.548556e-02	1.798463e-01	-1.984653e-02	-5.433583e-02	-2.741871e-01	4.010308e-02	2.235804e-02	-5.142873e-02	..
75%	139320.500000	1.315642e+00	8.037239e-01	1.027196e+00	7.433413e-01	6.119264e-01	3.985649e-01	5.704361e-01	3.273459e-01	5.971390e-01	..
max	172792.000000	2.454930e+00	2.205773e+01	9.382558e+00	1.687534e+01	3.480167e+01	7.330163e+01	1.205895e+02	2.000721e+01	1.559499e+01	..

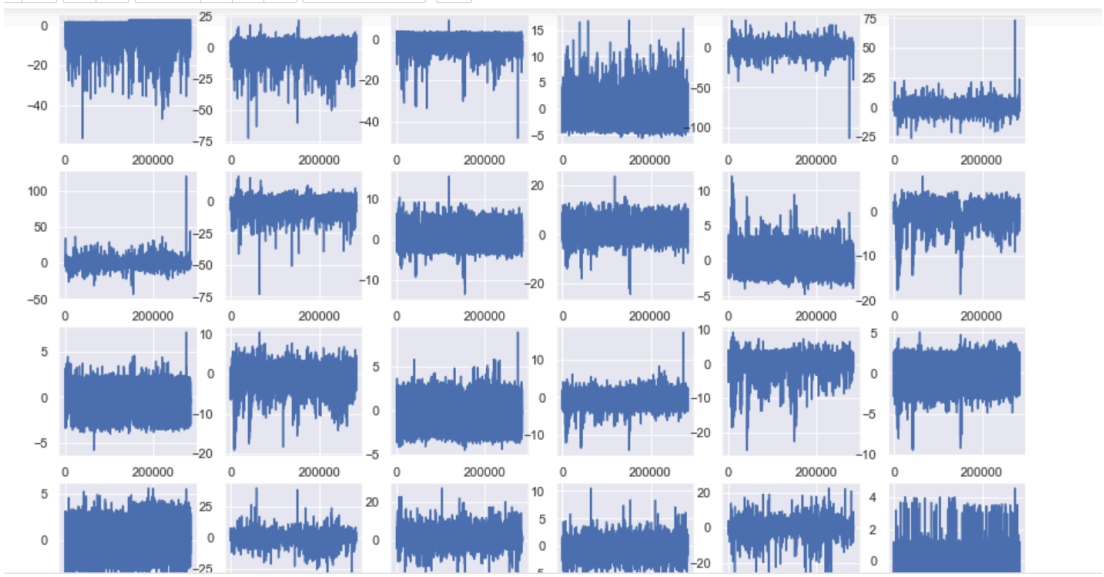
8 rows × 31 columns

```
class_names = {0:'Not Fraud', 1:'Fraud'}
print(df.Class.value_counts().rename(index = class_names))
```

```
Not Fraud    284315
Fraud           492
Name: Class, dtype: int64
```

```
fig = plt.figure(figsize = (15, 12))

plt.subplot(5, 6, 1) ; plt.plot(df.V1) ; plt.subplot(5, 6, 15) ; plt.plot(df.V15)
plt.subplot(5, 6, 2) ; plt.plot(df.V2) ; plt.subplot(5, 6, 16) ; plt.plot(df.V16)
plt.subplot(5, 6, 3) ; plt.plot(df.V3) ; plt.subplot(5, 6, 17) ; plt.plot(df.V17)
plt.subplot(5, 6, 4) ; plt.plot(df.V4) ; plt.subplot(5, 6, 18) ; plt.plot(df.V18)
plt.subplot(5, 6, 5) ; plt.plot(df.V5) ; plt.subplot(5, 6, 19) ; plt.plot(df.V19)
plt.subplot(5, 6, 6) ; plt.plot(df.V6) ; plt.subplot(5, 6, 20) ; plt.plot(df.V20)
plt.subplot(5, 6, 7) ; plt.plot(df.V7) ; plt.subplot(5, 6, 21) ; plt.plot(df.V21)
plt.subplot(5, 6, 8) ; plt.plot(df.V8) ; plt.subplot(5, 6, 22) ; plt.plot(df.V22)
plt.subplot(5, 6, 9) ; plt.plot(df.V9) ; plt.subplot(5, 6, 23) ; plt.plot(df.V23)
plt.subplot(5, 6, 10) ; plt.plot(df.V10) ; plt.subplot(5, 6, 24) ; plt.plot(df.V24)
plt.subplot(5, 6, 11) ; plt.plot(df.V11) ; plt.subplot(5, 6, 25) ; plt.plot(df.V25)
plt.subplot(5, 6, 12) ; plt.plot(df.V12) ; plt.subplot(5, 6, 26) ; plt.plot(df.V26)
plt.subplot(5, 6, 13) ; plt.plot(df.V13) ; plt.subplot(5, 6, 27) ; plt.plot(df.V27)
plt.subplot(5, 6, 14) ; plt.plot(df.V14) ; plt.subplot(5, 6, 28) ; plt.plot(df.V28)
plt.subplot(5, 6, 29) ; plt.plot(df.Amount)
plt.show()
```



11]:
```
from sklearn.model_selection import train_test_split
```

[7]:
```
feature_names = df.iloc[:, 1:30].columns
target = df.iloc[:1, 30: ].columns
print(feature_names)
print(target)
```

```
Index(['V1', 'V2', 'V3', 'V4', 'V5', 'V6', 'V7', 'V8', 'V9', 'V10', 'V11',
       'V12', 'V13', 'V14', 'V15', 'V16', 'V17', 'V18', 'V19', 'V20', 'V21',
       'V22', 'V23', 'V24', 'V25', 'V26', 'V27', 'V28', 'Amount'],
      dtype='object')
Index(['Class'], dtype='object')
```

[8]:
```
data_features = df[feature_names]
data_target = df[target]
```

12]:
```
X_train, X_test, y_train, y_test = train_test_split(data_features, data_target, train_size=0.70, test_size=0.30, random_state=1)
print("Length of X_train is: {X_train}".format(X_train = len(X_train)))
print("Length of X_test is: {X_test}".format(X_test = len(X_test)))
print("Length of y_train is: {y_train}".format(y_train = len(y_train)))
print("Length of y_test is: {y_test}".format(y_test = len(y_test)))
```

```
Length of X_train is: 199364
Length of X_test is: 85443
Length of y_train is: 199364
Length of y_test is: 85443
```

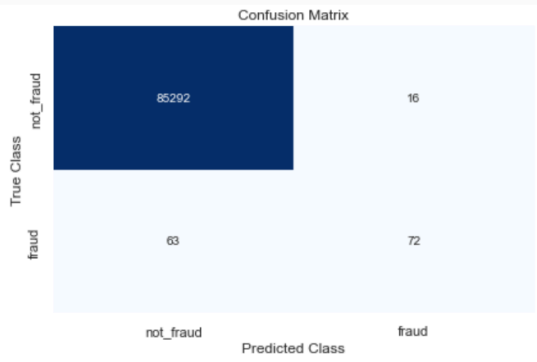
```
]: from sklearn.linear_model import LogisticRegression
   from sklearn.metrics import confusion_matrix
```

```
]: model = LogisticRegression()
   model.fit(X_train, y_train.values.ravel())
   pred = model.predict(X_test)
```

```
C:\Users\Manav Manan\anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:814: ConvergenceWarning: lbfgs failed to con
verge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
```

```
]: class_names = ['not_fraud', 'fraud']
   matrix = confusion_matrix(y_test, pred)
   # Create pandas dataframe
   dataframe = pd.DataFrame(matrix, index=class_names, columns=class_names)
   # Create heatmap
   sns.heatmap(dataframe, annot=True, cbar=None, cmap="Blues", fmt = 'g')
   plt.title("Confusion Matrix"), plt.tight_layout()
   plt.ylabel("True Class"), plt.xlabel("Predicted Class")
   plt.show()
```




```
: from sklearn.metrics import f1_score, recall_score
  f1_score = round(f1_score(y_test, pred), 2)
  recall_score = round(recall_score(y_test, pred), 2)
  print("Sensitivity/Recall for Logistic Regression Model 1 : {recall_score}".format(recall_score = recall_score))
  print("F1 Score for Logistic Regression Model 1 : {f1_score}".format(f1_score = f1_score))
```

```
Sensitivity/Recall for Logistic Regression Model 1 : 0.53
F1 Score for Logistic Regression Model 1 : 0.65
```

Conclusion: By following these steps and leveraging the power of machine learning and data science, you can develop a sophisticated credit card fraud detection system that significantly reduces the risk of fraudulent transactions and enhances the security of credit card usage.